CS172 Computability & Complexity (Spring'09)

Instructor: Mihai Pătraşcu GSI: Omid Etesami

$\boxed{\text{Problem Set 6}}$ *Due: Thursday, March 12*

Homework policy: As in homework 5; please see website.

1. Write code that will evaluate mathematical expressions involving the symbols $+$, $-$, $\star$ (multiplication), $/$ (division), and $\wedge$ (exponentiation). You may assume that the expression only contains single-digit numbers.

Your code should be based on following context free grammar defining the language of such expressions:

$$\begin{aligned} S &\rightarrow S+T \mid S-T \mid T \\ T &\rightarrow T*U \mid T/U \mid U \\ U &\rightarrow V \wedge U \mid V \\ V &\rightarrow (S) \mid -V \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

You code should read one line from the standard input, containing such an expression. Then, it should output one line containing the answer. Make all computations with an integer data type. (Test cases are promised not to contain expressions like $2 \wedge (-1)$ or $1/2$, which would evaluate to non-integral values.)

Make sure your code passes all the tests found at: `http://inst.eecs.berkeley.edu/~cs172/sp09/hw6-p1.txt`. Email the source code (in C, Java, or Pyhton) to `mip@alum.mit.edu` as an attachment to an email with the subject `CS172-HW5-P1`. Grading will be done automatically so please follow I/O and email specifications carefully.

2. [Hard] Solve problem 1 without using recursive functions. That is, all your parsing should be done in one function that never calls itself. Test and submit as above.

3. Here are a few basic exercises for context-free grammars.

- Remember that the language $L = \{w\, w^R \mid w \in \{0,1\}^\star\}$ is not regular (as we showed using the pumping lemma). Show that L is a context-free language (i.e. write a context free grammar for it).
- Write an unambiguous context-free grammar for the set of strings over $\{0,1\}$ that contain more ones than zeros. Explain why your grammar "works" and is not ambiguous.
- Consider the following context-free grammar that tries to capture if-then-else statements in a regular programming language:

$$\begin{aligned} S &\rightarrow \texttt{PRINT hello} \mid T \mid U \\ T &\rightarrow \texttt{IF condition THEN } S \\ U &\rightarrow \texttt{IF condition THEN } S \texttt{ ELSE } S \end{aligned}$$

Prove by example that the grammar is ambiguous. Make it unambiguous.

4. Consider the following communication problem, which we call Median:

- Alice receives $A[1 \dots n]$, where each $A[i] \in \{0, \dots, n^2\}$.
- Bob receives $A[n+1 \dots 2n+1]$, where each $A[i] \in \{0, \dots, n^2\}$.
- their goal is to find the median of the array $A[1 \dots 2n+1]$.

In class, we considered the communication problem Indexing:

- Alice receives $x \in \{0,1\}^n$.
- Bob receives $y \in \{1, \dots, n\}$.
- their goal is to find x_y.

We proved that if Indexing is solved by sending a single message from Alice to Bob, this message must have size $\Omega(n)$ bits. Using this lower bound for Indexing, prove that: if Median is solved by sending a single message from Alice to Bob, the message must have size $\Omega(n)$ bits.

Hint: If you are having difficulties, see `http://inst.eecs.berkeley.edu/~cs172/sp09/hw6-p4.txt`

5. Show that Median can be solved with $O(\lg n)$ messages, each of $O(\lg n)$ bits.
Hint: If you are having difficulties, see `http://inst.eecs.berkeley.edu/~cs172/sp09/hw6-p5.txt`

6. [Hard] Show that Median can be solved with $O(\lg n)$ messages, each of $O(1)$ bits.

7. [Hard] Consider a model of computation with an *input* and *output* stream. At each step, the algorithm may either read the next character of the input, or output one more character to the output stream. The output stream is "append only," i.e. once a character is written, it may never be modified again. It is also impossible to "seek" in the output stream (each write adds a character at the end of the output stream).

The input stream will contain a number written in base 3, starting from the most significant digit. The number will have n digits ("trits"); assume n is known in advance. You goal is to write the base 2 representation of the number on the output stream. Your streaming algorithm should use only $O(\lg n)$ bits of space.

The story: In this problem, you are essentially implementing a compression technique known as "arithmetic coding." Here is an overview of arithmetic coding for your reading pleasure.

Consider a distribution $\mathcal{D}$ over support Σ; let $p(\cdot)$ be the probability density function. Assume that we receive a text of n characters, each drawn independently from $\mathcal{D}$. Our goal is to compress this text.

In class, we claimed that the any compression algorithm cannot compress the text to less than $n \cdot H(\mathcal{D}) = n \sum_{x \in \Sigma} p(x) \log_2 \frac{1}{p(x)}$ bits on average. It is known that Huffman coding will use at most $H(\mathcal{D}) + 0.419$ bits on average per character. Thus, it uses $n \cdot H(\mathcal{D}) + O(n)$ bits in total. On the other hand, arithemtic coding achieves $n \cdot H(\mathcal{D}) + O(1)$ bits on average, which is essentially optimal.

Here is how arithmetic coding works. The empty text is viewed as the interval $[0,1]$. To represent the first character, break the interval into Σ pieces, where the piece corresponding to x has length proportional to $p(x)$. To represent the second character, break each interval into subintervals, and so on. At the end, there are Σ^n subintervals, of different lengths — the interval corresponding to text $x_1 x_2 \dots x_n$ has length $\prod_{i=1}^{n} p(x_i)$.

The encoding of the text is an arbitrary number in the interval. Since intervals are disjoint, this uniquely identifies the interval and thus the text.

How many bits does a fractional number need to have to hit such a tiny interval? (I am taking about number of the form "0.abcd") Well, if an interval has length greater than 2^{-k}, there exists a k-bit number inside it (think about it). So to represent text $x_1 x_2 \dots x_n$, I will need roughly $\log_2 1/\prod_i p(x_i)$ bits in my number. But that is just $\sum_i \log_2 \frac{1}{p(x_i)}$, so it's like x_i contributed $\log_2 \frac{1}{p(x_i)}$ to the encoding. This is essentially what this entropy business means.

The only question is how to efficiently generate the binary number that hits the interval (the trivial algorithm needs ridiculously large precision for floating point operations). The solution is the same as in the problem of converting base 3 to base 2. (Think about it.)

8. [Open] In class we showed that norm estimation requires $\Omega(1/\varepsilon^2)$ bits of communication, if there is a single message from Alice to Bob. Unlike Indexing and Median, we do not know a better result if we allow multiple messages. Prove that $\Omega(1/\varepsilon^2)$ bits of communication are necessary even if Alice and Bob may exchange an arbitrary number of messages.

This is an open problem (i.e. nobody knows how to do it yet). Mihai has worked on it, and he will be glad to listen to you if you have ideas.